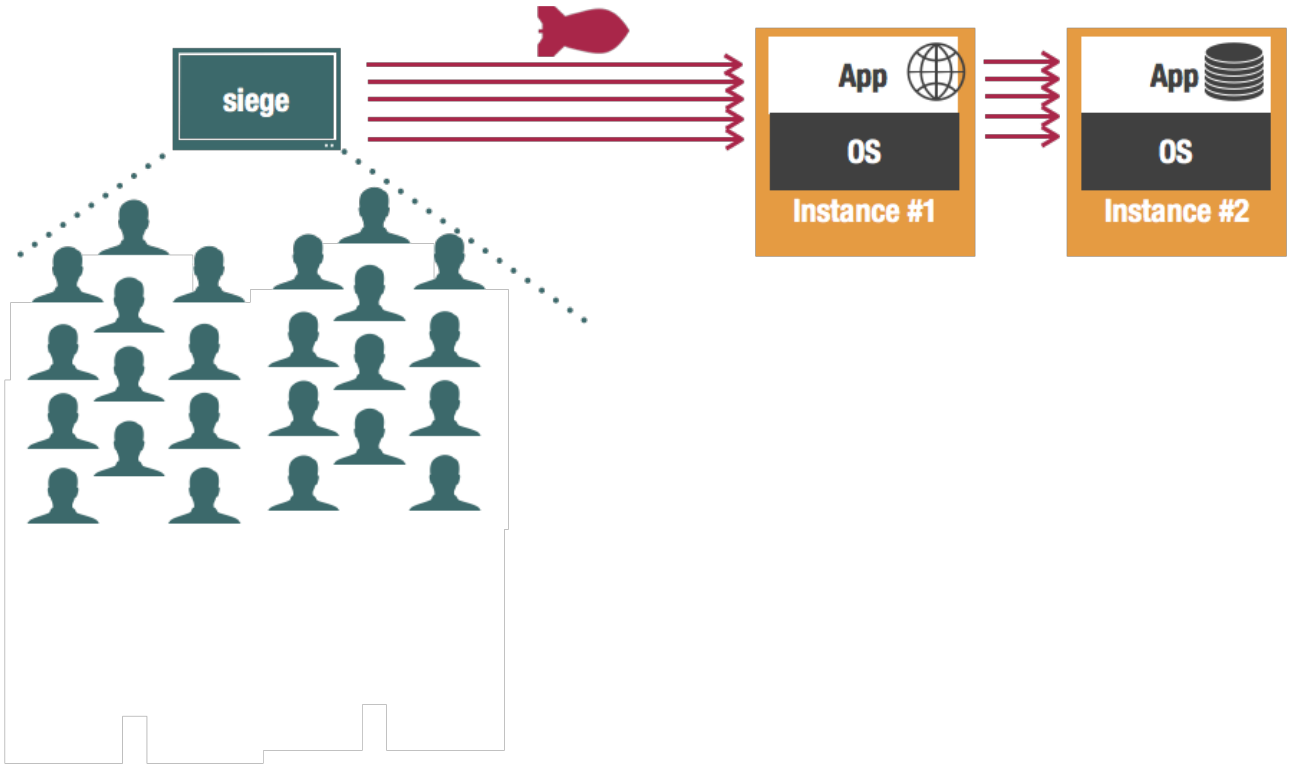
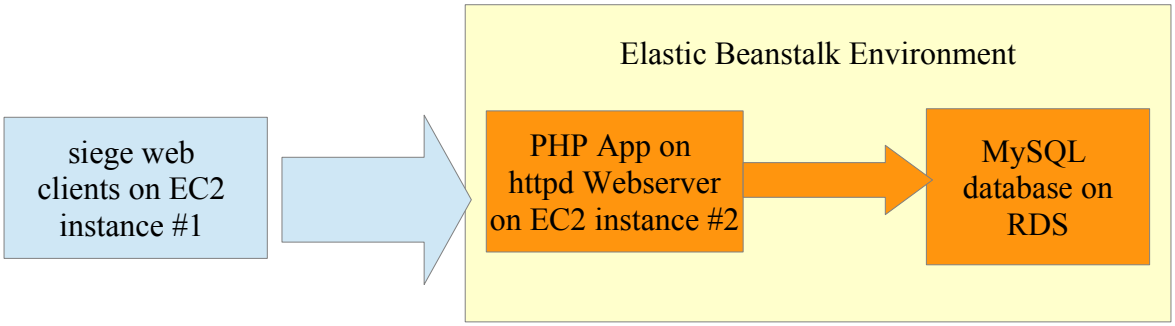
Table of Contents



I. Prepare before Activity

Goal: In the same AZ



A. Set up "siege" your web client for load testing to run on your Amazon EC2 instance (Amazon Linux)

Get an EC2 Amazon Linux instance (micro instance free tier). Select an AZ, and note which one you use. Install siege. Refer to the steps from Activity 2 for this part.

B. Prepare php web application

Download all the web app files for activity 3 from myCourseville onto your notebook. We will use this for part II and III.

There are 3 files: index.php, styles.css, logo_aws_reduced.gif.

Make sure the matrix size is 128

```
$size = 128;
```

Edit database code to index.php

For PaaS, edit your index.php to have the following statement verbatim after the comment:

```
<!-- add mysql connection here -->

<?php

$conn =   new mysqli($_SERVER['RDS_HOSTNAME'], $_SERVER['RDS_USERNAME'], $_SERVER['RDS_PASSWORD'], 'ebdb');
```

II. Activity 3 PaaS Auto-scaling
A. Deploy app using Elastic Beanstalk

Use the 3 web app files that you have edited (index.php, styles.css, logo_aws_reduced.gif).

Zip them into a format that Beanstalk will recognize. Be **careful** here as Beanstalk is particular about this **format**.
https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/applications-sourcebundle.html

Go to the AWS Elastic Beanstalk console, and launch a new PHP environment.

Upload your zip file. Wait for deployment to complete.

B. Set up MySQL database

In the Elastic Beanstalk dashboard, select your newly created environment.

Select "Configuration". In the "Database" panel, select Modify.

Create a new MySQL RDS instance (micro instance free tier) here.

Apply the changes to the environment. You will see beanstalk create a database called "ebdb" for you.

Do not forget your username and password.

Connect to the database using MySQL workbench, and create a table called user with no data inside, containing 2 fields: id and username. If you cannot connect, check your database's security group.

user

id (int)	username (varchar 63)

C.Test that your app works

Test the URL for your environment, for example,
http://Default-Environment.gt3p835vxb.ap-southeast-1.elasticbeanstalk.com

Check your table in MySQL and the webpage to see if your database code works.

Run siege against your web application

```
$ siege -c10 -d1 -r1 http://default-environment.gt3p835vxb.ap-southeast-
1.elasticbeanstalk.com/
```

Look at the results from siege. Look at the monitoring panel to see the resources in your environment. If your environment degrades/fails healthchecks, reconfigure the healthcheck settings

as discussed in class.

D. Assess application baseline performance

Use siege to test the baseline performance of your web app deployed using PaaS. Measure and plot the

- (average) response time and
- (average) throughput

as a function of offered load (# of client requests/second). Show the inflection point on both curves and indicate how you would map this to the appropriate system resources (performance metrics) to use to trigger auto-scaling for PaaS in your report.

E. Set up your auto-scaling environment

Configure your scaling policy under "Configuration". In the "Capacity" panel, select Modify.
Change the environment type to "Load Balanced."
Create your auto-scaling policy for

- max of 4 instances
- select the same AZ as your siege EC2 instance
- when to activate (add) new instances
- when to remove instances

Select only 1 AZ, using the same one as your siege instance.

Make sure that your scaling policy makes sense with respect to your baseline application performance. Use a suitable/realistic resource to trigger autoscaling as you would for any real application.

Use siege from your EC2 instance to trigger scaling up and scaling down, adjusting the values for the flags (-c, -d, -r) to trigger auto-scaling events.

```
$ siege -c10 -d1 -r1 http://default-environment.gt3p835vxb.ap-southeast-
1.elasticbeanstalk.com/
```

F. Write your Lab report

1. From part D, show the inflection point on both of your curves and explain how you map this to the appropriate system resources (performance metrics) to use to trigger auto-scaling for PaaS.
2. Describe the scaling policy you configured to scale up and scale down.
3. How did you run siege (what options) to trigger scaling up and scaling down?
4. Was the scaling up and scaling down behavior consistent with your scaling policy?
5. Include screenshots from Elastic Beanstalk showing the number of instances running and screenshots of resource monitoring data for your environment in your report to confirm that you successfully triggered auto-scaling according to your configured policies.

III. Activity 3 IaaS Auto-scaling

Check that the following items from Activity 2 and 3 are still around. If not, redo them.

- EC2 instance with siege already installed

- Set up a PHP web app and a database instance on EC2 in the same AZ as the siege instance. Make sure that your PHP code refers to your database instance on EC2. You may also choose to use the RDS instance from your PaaS application instead of a separate database instance on EC2.

A. Assess your application's baseline IaaS performance

Assess your application's baseline IaaS performance. We actually already did this for Activity 2, so you do not need to repeat this if your work from Activity 2 is usable. However, if you results from Activity 2 are unclear, redo this part.

Use siege to test the baseline performance of your web app deployed using IaaS. Measure and plot the

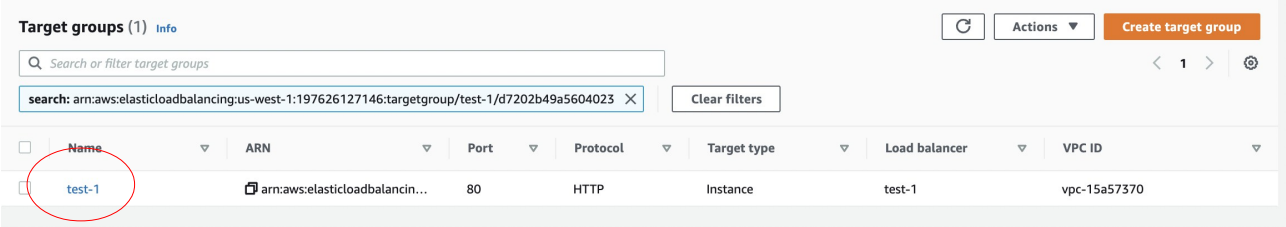
  - (average) response time and
  - (average) throughput

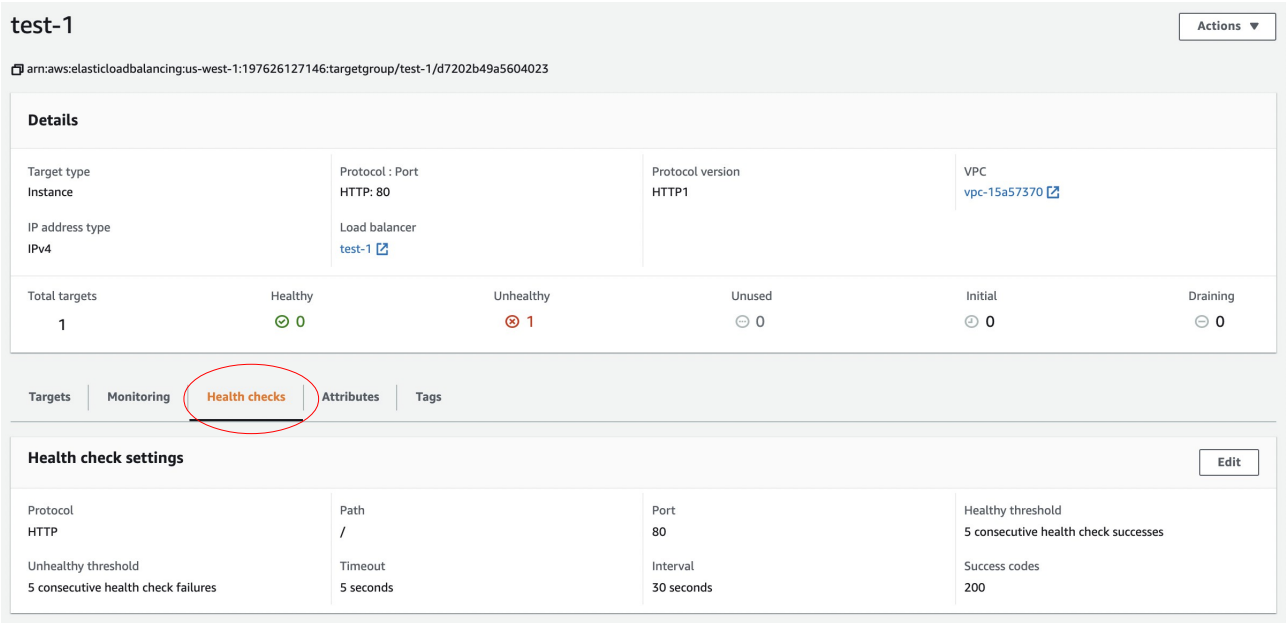
as a function of offered load (# of client requests/second), similar to what you did for PaaS. Show the inflection point on both curves and indicate how you would map this to the appropriate system resources (performance metrics) to use to trigger auto-scaling for IaaS.

B. Set up your IaaS auto-scaling environment

1. Create an ami and test that it works.
   1. Create an ami from your webapp EC2 server called **phpiaasapp**.
   2. Test that your **phpiaasapp** ami works by launching a new EC2 instance that uses your ami in the same AZ as your RDS instance, and call the instance ***"testami"***. Use your browser to see if the webpage is up.
   3. Terminate the ***"testami"*** instance.
2. Create a launch template from your **phpiaasapp** ami, called **phpiaasapplaunch** and set the instance type to the free tier instance in your region (e.g., t2.micro). Make sure your security group is set up correctly to enable web traffic. **Hint:** Launch templates have built-in version control. You might need to change the version selected for the auto-scaling group if you modify the template after creation.
3. Create an Auto-Scaling Group
   1. Create a new auto-scaling group called **phpiaasgroup.**

2. Associate your **"phpiaasapplaunch"** launch configuration to this group.
3. You need to select two subnets from two AZs. Make sure one of them is in the **same AZ (subnet)** as your RDS instance.
4. In the "Integrate with other services" tab, select "Attach to a new load balancer" and configure the application elastic load balancer (ELB). Make sure it is Internet-facing.
5. Create your auto-scaling policy for
    - scaling between 1-4 instances,
    - when to activate (add) new instances, and
    - when to remove instances.
    - Make sure that your scaling policy makes sense with respect to your baseline application performance. Use a suitable/realistic resource to trigger autoscaling as you would for any real application.
6. Test that your IaaS auto-scaling web app works by using your browser and going to your ELB's URL. All future client web requests will use the URL associated with the ELB instead of any web app EC2 instances. Why?
7. Configure your ELB healthcheck by going to the Target Group menu and select the corresponding target group. Then go to the healthcheck tab to configure it to match your application.





C. Experiment with siege to trigger events to

add instances and remove instances

Use siege from your EC2 instance to trigger scaling up and scaling down, adjusting the values for the flags (-c, -d, -r) to trigger auto-scaling events.

```
$ siege -c10 -d1 -r1 http://elasticloadbalancername/index.php
```

Use the name of the ELB, not your web app EC2 instances!

D. Experiment with fault tolerance

While you have at least 2 instances running, kill one and run siege/access the URL from your browser to observe the impact on clients.

Kill your remaining instance, observe what happens, and run siege/access the URL from your browser again to observe the impact on clients.

Answer these questions in your lab report (Q#6): Is Amazon able to launch new instances to replace the one(s) you killed? When the new instances are launched, are your web clients (browser or siege) able to access the web application like normal?

Provide evidence of your observation (screenshots, logs).

E. Write your Lab report

1. Consider observed **baseline** performance in terms of response time and throughput as a function of offered load (# of client requests/second) when you deploy the same web app to to PaaS vs. IaaS. Double-check to see if they are deployed on the same actual instance sizes and check the AZ.
   - 1.1. Discuss the differences in **baseline** performance in terms of response time and throughput as a function of offered load (# of client requests/second), and conclude which provides better performance?
   - 1.2. Are there any reasons that you think would sensibly explain the performance differences?
2. For IaaS, what scaling policy did you use to scale up and scale down?
3. How did you run siege (what options) to trigger scaling up and scaling down?
4. Was the scaling up and scaling down behavior consistent with your scaling policy?
5. Include screenshots from EC2 monitoring data in the report to make sure I can see that you did trigger auto-scaling according to your configured policies.
6. Consider what Amazon did when you terminated your EC2 instance(s) in part D (fault tolerance).
   - 6.1. Is Amazon able to launch new instances to replace the one(s) you killed?
   - 6.2. When the new instances are launched, are your web clients (browser or siege) able to access the web application like normal? Provide evidence of what your web clients see.
7. Discuss the differences between **configuring auto-scaling in IaaS** vs. **auto-scaling PaaS.** What are the strengths of each configuration approach?
8. Discuss the differences between IaaS vs. PaaS. What are the strengths of each approach?